



CONSULTANCY

Unifying Data Delivery Systems Through Data Virtualization

A Whitepaper

Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

October 2018

Sponsored by

fraXses

Copyright © 2018 R20/Consultancy. All rights reserved. All company and product names referenced in this document may be trademarks or registered trademarks of their respective owners.

Table of Contents

| | | |
|----------|---|----|
| 1 | Executive Summary | 1 |
| 2 | The Siloed World of Data Delivery | 2 |
| 3 | What To Do? | 5 |
| 4 | Data Virtualization in a Nutshell | 6 |
| 5 | The Unified Data Delivery Platform | 9 |
| 6 | Requirements for Data Virtualization Technology | 11 |
| 7 | The fraXses Data Virtualization Platform | 14 |
| 8 | Closing Remarks | 18 |
| | About the Author | 20 |
| | About fraXses, Inc. | 20 |

1 Executive Summary

Big Data or Big Data Usage? – Many regard *big data* as the biggest change in the IT industry of the last decade. However, big data is still just data. Like ordinary data, big data consists of words, codes, numbers, dates, images, videos, and text. The big data trend did not introduce a new type of data. Big data systems process, store and analyze more of the same data. Evidently, to efficiently support these massive quantities of data, new technology was needed. Having to process and store more of the same data hardly classifies as the biggest change in the industry.

The real big change of the last decade is related to how organizations use data. Organizations use data more intensively, more widely, and in many more forms. Organizations have started their *digital transformation journeys* and are striving towards becoming increasingly *data-driven organizations* and this has led to this drastic change in *data usage*. Traditional forms of data usage, such as reporting and dashboards, are not sufficient anymore. There is a growing need to exploit the investment made in data by deploying almost every form of data usage. Big data usage is the biggest change.

Big data is not the biggest change in the IT industry but big data usage is.

This big data usage change is directly related to the fact that organizations have started to understand the potential value of data. They recognize that data by itself has no value. It's not like money. Money has value even if you don't use it. Data, on the other hand, has value only when it's used. Therefore, organizations want to use their data in all possible ways.

Data has value only when it's used.

New Forms of Data Usage a Reality – There was a time when data usage meant reporting and some simple forms of analytics. The available forms of data usage have grown enormously. Here are some of the more popular new ones:

- Business analysts using self-service BI capabilities for fulfilling their reporting needs.
- Data scientists using statistical and deep learning techniques to discover trends and patterns in the data.
- Customers who initiate simple forms of analysis through a mobile app.
- Operational applications with embedded BI components that show historic data or results of forecasting models.
- Automated decisions based on BI decision models created with artificial intelligence techniques.
- Analysis of streaming data.
- Supplier- and customer-driven BI.
- Applied artificial intelligence programs to analyze text, images, and video.

Myriad of Data Delivery Systems – In their hurry and enthusiasm to implement the new forms of data usage, organizations have developed a myriad of isolated systems that deliver data to some users. They have developed a labyrinth of *data delivery systems*. For example, a *data warehouse* is developed to support the more traditional forms of reporting and dashboarding; a Hadoop-based *data lake* is under construction to support data science and other forms of investigative analytics; an old managed file transfer system still delivers files to legislators, customers, and

Organizations have developed a labyrinth of data delivery systems.

suppliers; a new API gateway with which mobile apps can access centralized data is operational; and they have their new Kafka-based streaming apps to deliver data live.

Most of the time, these siloed data delivery systems share no technologies, no specifications, no development groups, no data security and data privacy specifications, and no solutions. They share nothing. They are developed, operated, and maintained by different developer groups. This isolated approach of data delivery development has the following drawbacks:

- Decreased time to market of reports and analysis
- Decreased report consistency
- Duplication of data
- Decreased transparency
- Higher development costs

The Whitepaper – When an organization wants to professionalize and streamline their data delivery systems, they must unify these systems. The silos have to come down. Evidently, this is not a simple exercise. This whitepaper describes a new architecture for developing one *unified data delivery platform*. One that potentially supports all forms of data usage, from the simplest reports to the most advanced forms of analytics. The key technology for such a unified platform is *data virtualization*. The whitepaper lists the minimal requirements for data virtualization technology to support this new architecture. The whitepaper closes with a description of the *fraXses* data virtualization platform and how it meets the requirements for a unified data delivery platform.

Data delivery systems must be combined into one unified data delivery platform.

2 The Siloed World of Data Delivery

Transformation Specifications – Regardless of the nature of a data delivery system, whether it's a data warehouse, a data lake, or a data marketplace, they have certain things in common: they all extract data from source systems, subsequently they transform that data, and finally the transformed data is delivered in some form to a data consumer; see Figure 1. Note that a data consumer can be an internal business user, a Java app running on a mobile device, or a data scientist.

The source systems are not controlled by the data delivery systems. To transform source data, numerous *transformation specifications* are needed, including aggregation, filtering, cleansing, data security, and integration specifications. For example, in a data warehouse environment these specifications are implemented as lines of programming code in ETL programs, database stored procedures, semantic layers, or simple in hand coded pieces of Java code. Whereas in a data lake they may end up as code in MapReduce, Python, or R code.

Transformation specifications are tremendously valuable.

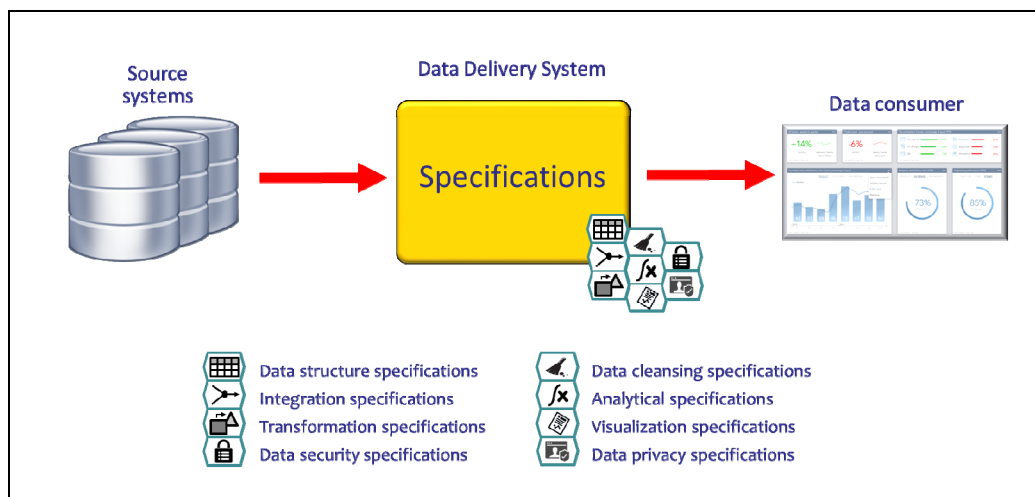




Figure 1 What all data delivery systems have in common is that they extract, transform, and deliver data to data consumers.

These transformation specifications are tremendously valuable. In a way they are the *intellectual property* of an organization. They describe precisely how an organization wants to process and consume data. The specifications are developed specifically for them to transform all the data they have into forms and shapes required by the data consumers.

Data Delivery Silos – With respect to implementing the specifications, in many organizations the wheel is reinvented over and over again. For example, if a data warehouse and data lake access the same source systems, the specifications to integrate this data are implemented twice. In the data warehouse a professional ETL tool is probably used to implement the integration specifications, whereas in the data lake that same specification is implemented using Python code, or a different ETL product, such as Apache Sqoop. Something similar may happen with data security and filtering specifications.

This approach leads to *siloed data delivery systems*. Each silo contains its own implementation of the specifications. For example, Figure 2 shows an environment consisting of three data delivery systems in which two specifications are developed three times. In this example, data from two source systems (for example, invoice data and customer data) must be integrated (see the  icon), and some complex calculation on that data must be performed (see the  icon). Firstly, In the data warehouse both specifications may be implemented with an ETL tool. The result of the integration and calculation are both stored in the data warehouse. Secondly, a data scientist using the data lake may be interested in integration of the same invoice and customer data and in using the same calculation. The code for the integration and the calculation may be developed with R. That code does not run on a server but on the client side. Thirdly, a mobile app may want to show customer and invoice data. The code to integrate these data sources and to perform the calculation are probably developed in Java and runs on a server. The result is three different, independent implementations of the same specifications.

With siloed data delivery systems each silo contains its own implementation of the specifications.

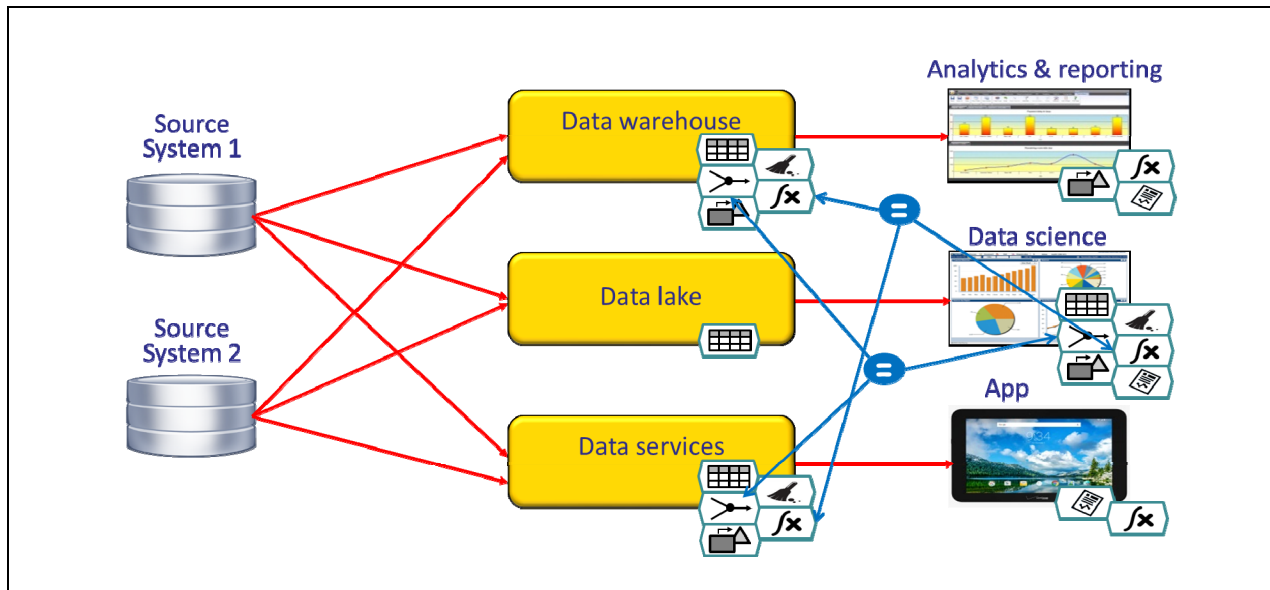


Figure 2 An environment consisting of three data delivery systems in which specifications are regrettably replicated.

The Disadvantages of Siloed Data Delivery Systems – This siloed approach has some severe drawbacks:

- **Decreased time to market:** In every data delivery system, specifications are implemented anew. In other words, the wheel is reinvented over and over again by the developers. This means that no one benefits from reusing existing specifications. This slows down development and thus negatively influences the time to market for every new form of data usage.
- **Decreased report consistency:** It is a gargantuan effort to keep all the different implementations of the specifications in synch with each other. If a specification changes, how can an organization guarantee that all the implementations are updated accordingly and that they are all updated consistently? And if the specifications are inconsistent, reporting results will be inconsistent.
- **Duplication of data:** In each data delivery system data is copied from the source systems to their data stores. Additionally, inside each data delivery system data is duplicated for performance or other reasons. Looking at all the data delivery systems, one data fact may be stored tens of times.
- **Decreased data quality:** The more often data is copied, the greater the chance that the copied data contains a discrepancy. It's similar to that game in which at one end of a line of people someone whispers a sentence in the ear of his neighbor, that neighbor whispers the sentence to his neighbor, and so on. The longer the chain of people, the greater the chance that the sentence ends up completely mangled at the other end of the line.
- **Decreased transparency:** All the copying of data, all these different implementations, and all the different technologies in use result in a highly complex and non-transparent data delivery system. Transparency has become a key requirement for business users today. They want to know whether they can trust the data they base their decisions on.

- **Higher development costs:** Due to the isolated nature of the siloed data delivery systems there is no or limited sharing or reusing of implementations. This increases development costs unnecessarily.

Why don't the developers see this? Most developers work on only one data delivery system. They are not even aware of what's being developed in the other delivery systems. They don't see that in another data delivery system the same technical problems are being solved. They don't see the big picture, they only see their own silo.

Data delivery systems share specifications, but they don't share the same implementations.

To summarize this section, the key issue is that all the data delivery systems share specifications, but they don't share the implementations of those specifications; see Figure 3.

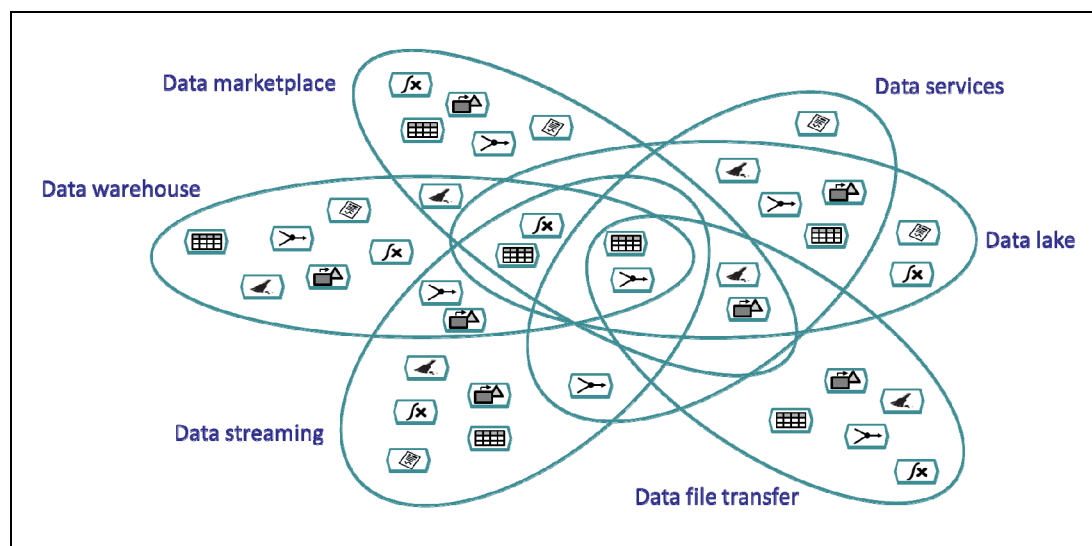


Figure 3 Data delivery systems share specifications, but they don't share the same implementations.

3 What To Do?

If organizations don't solve this puzzle they will eventually run aground. Implementing new specifications, changing them, and adding new forms of data usage, will become increasingly time consuming and expensive. What is needed is one integrated and *unified data delivery platform*. One in which specifications are implemented once and shared amongst data delivery systems. This requires that in the overall architecture the specifications form the key building block, not the data stores. Figure 4 displays such an architecture in which data extraction and data delivery are streamlined.

To develop such an architecture a key technology is needed that supports a centralized definition of specifications and is capable of supporting a wide range of data consumers, from JSON/REST based APIs to data scientists. A technology that has proven itself numerous times for this is *data virtualization technology*. For those not familiar with data virtualization, the next section contains a short introduction.

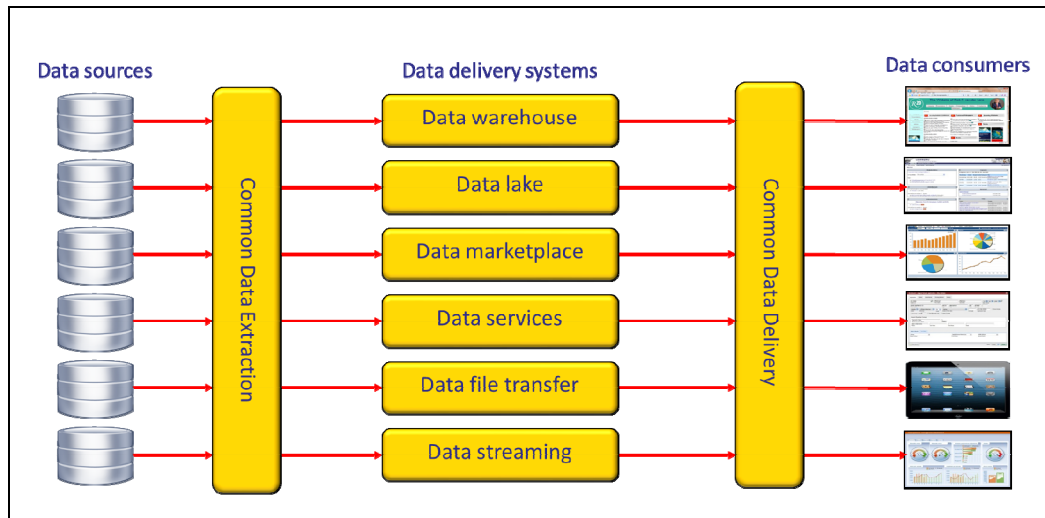


Figure 4
Organizations need a unified data delivery platform.

4 Data Virtualization in a Nutshell

Introduction — Data virtualization is a technology for integrating, transforming, and manipulating data that’s available in all kinds of data sources and for presenting all that data as one *logical database*. When data virtualization is applied, an *abstraction layer* is provided that, for applications, hides most of the technical aspects of how and where data is stored; see Figure 5. Because of this layer, applications don’t need to know where all the data is physically stored, whether the data is on a windows server or in the cloud, how the data should be integrated, where the database servers run, how to insert and update the data, what the required APIs are, which database language to use, and so on. When data virtualization is deployed, to every application it feels as if one large database is accessed.

With data virtualization data consumers are decoupled from data sources.

Applications can use different interfaces to access data, including SQL, SOAP, and REST. When, for example, SQL is used to access data stored with Hadoop, the data virtualization platform transforms SQL into code to access Hadoop. Data virtualization platforms can also change the structure of the source data, if necessary. If data comes from multiple systems, the data virtualization platform integrates the data and presents a unified result to the application.

Definition of data virtualization¹: *Data virtualization is the technology that offers data consumers a unified, abstracted, and encapsulated view for querying and manipulating data stored in a heterogeneous set of data stores.*

¹ Rick F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann, 2012.

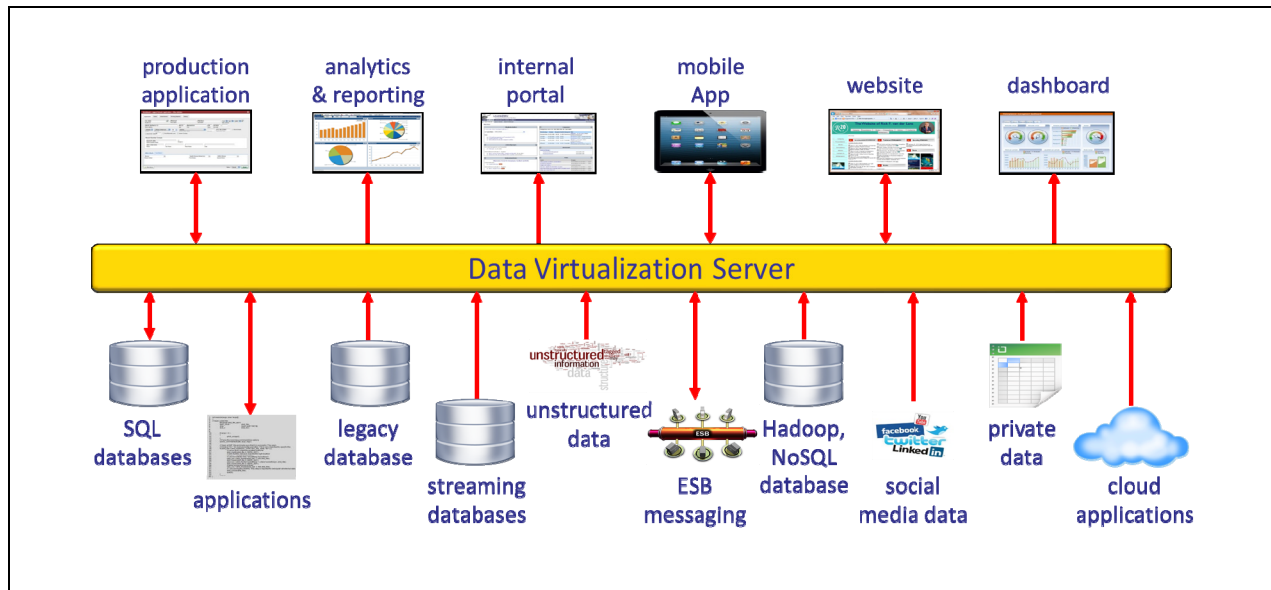


Figure 5 Data virtualization platforms ensure that a heterogeneous set of data sources looks like one logical database to the data consumers. The data virtualization platform is responsible for data integration.

The Key Advantages of Data Virtualization – The key advantages of data virtualization when deployed to retrieve data are the following:

- **Easy Data Access:** Some data sources support complex concepts and offer only highly technical interfaces. This makes the data in such systems difficult to access. With data virtualization platforms, users and applications can use easier-to-use languages, such as SQL, to access data. All the technical concepts and interfaces are hidden.
- **On-Demand Data Transformation:** When users retrieve data via a data virtualization platform, that data is retrieved from the source systems and transformed to the right SQL table structures *live*. So, when data is accessed from a data source, it's not first copied to and stored in another database or file, but it's streamed live to the users instead. In other words, data virtualization supports on-demand data transformation.
- **On-Demand Data Integration:** In contrast to ETL tools, data virtualization platforms integrate data available from different sources live. As with on-demand data transformation, there is no need to store the integrated result first before it can be used.
- **Real-Time Data:** Because source data can be accessed directly and there is no need to access a database containing a copy of the data, applications see real-time data. Users see the current state of the data and not yesterday's data.
- **Read/Write Interface:** With data virtualization, source data is accessed directly. This allows applications to read as well as insert and update data.
- **Increased Agility:** Data virtualization increases agility, because developers don't have to use the complex and tedious languages supported by the data sources. In addition, because the applications are decoupled from the data sources, it's easier to make changes to the source

systems without impacting the applications. All this dramatically improves the agility of a data delivery system.

- **Improved Time-to-market:** If data is easier to access, integrating existing data sources and plugging in new data sources is less time-consuming. In other words, it improves the time-to-market allowing organizations to react faster.
- **Simple Architecture:** Because data virtualization deploys on-demand data integration, no additional databases have to be managed and maintained.
- **Centralized Data Security:** Because on-demand integration is deployed, all the security mechanisms protecting the data sources still apply. Additionally, within the data virtualization platform fine-grained data security rules can be defined centrally on any data source.

Virtual Tables – The core building block of a data virtualization platform is called a *virtual table* (or *data object* or *view*). The definition of a virtual table specifies which *transformation operations* must be applied to source data when it's retrieved by users. Examples of transformation operations are aggregations, filters, joins, calculations, and concatenations. Through these virtual tables, source data can be transformed into whatever the business users wish.

When users access data through the virtual tables, all the transformation operations are applied to the source data. For example, if a virtual table contains a filter and aggregation operation, when users access it, both transformation operations are applied to the data before it is returned to them. In other words, the operations making up the definition of a virtual table, are applied *on demand*.

Virtual tables form the building blocks that describe how data must be transformed.

Layers of Virtual Tables – A best practice is that virtual tables are layered; see Figure 6. Each layer of virtual tables has a certain task. The bottom layer of the virtual tables is commonly responsible for extracting the data from the data stores. The middle layer takes care of the integration of data and if required the cleansing of the data. In the top layer the virtual tables have a structure that meets the requirements of the data consumers. Together, the three layers transform the data coming from the data sources to a form required by the data consumers. In other words, the transformation specifications described in Section 2 are defined in the virtual tables.

For different forms of data usage, different virtual tables are defined. For example, a data scientist may want to see the data differently from a more traditional business user. The former may want to have all the required data organized as one wide table, while the latter wants it organized as a star schema.

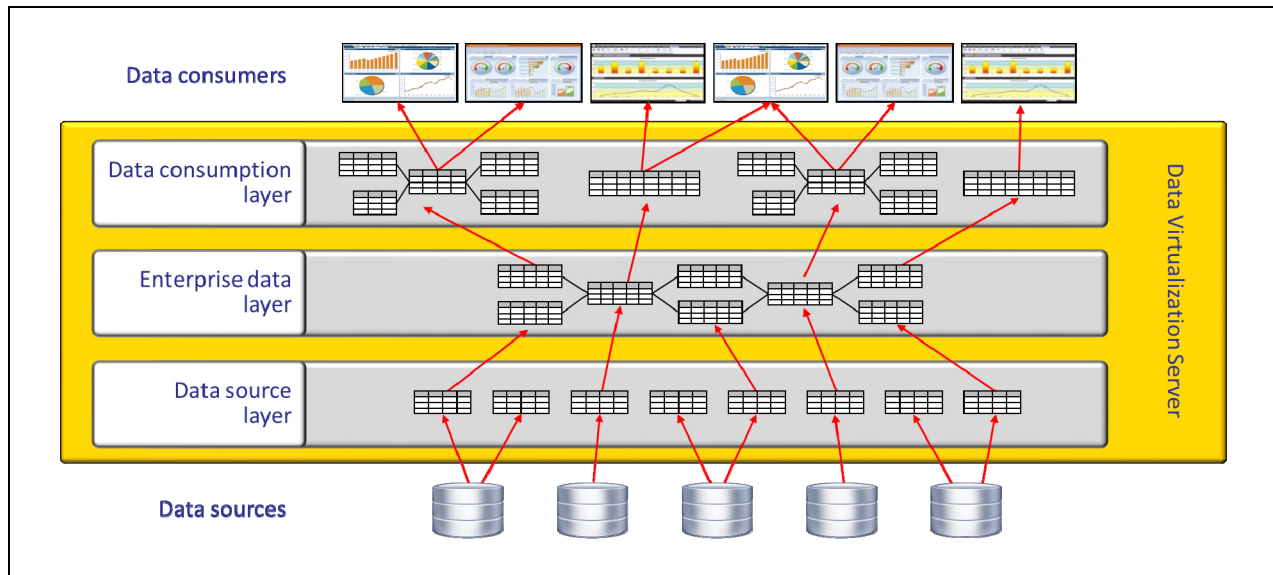


Figure 6 Layers of virtual tables.

Summary – With data virtualization, all the data, regardless of where it’s store, how it’s stored, and how it must be accessed, can be presented as one large, logical database that can be accessed by any data consumer through the interface they need and prefer.

5 The Unified Data Delivery Platform

The High-Level Architecture – Many roads lead to Rome just as there are many different ways to develop a unified data delivery platform. The one presented in this whitepaper places data virtualization technology at the center. Figure 7 contains a high-level representation of the architecture of the unified data delivery platform.

In this architecture data from all data sources is first copied to a *data hub*; structured and unstructured data. The data hub contains unprocessed data, it also houses historic data. It may deploy different storage technologies for different types of data. Next, some of the data is copied to a *data warehouse*. In this step data is processed as minimally as possible. This implies that this data warehouse is not designed in the way that most traditional data warehouses are designed. In the latter data is commonly integrated, filtered, aggregated, standardized, and so on. The goal will be to implement as many of the transformation specifications within the data delivery module and not in the process that moves data from the data hub to the data warehouse.

The *data delivery module* is the heart of the architecture. This is where most of the transformation specifications are implemented. The technology used here is data virtualization. All the data consumers use this data delivery module to get access to data. Depending on the requirements, data is retrieved from the data warehouse, from the data hub, or directly from the source systems. Inside the data delivery module, layers of virtual tables are defined to hide the data warehouse, the data hub, and the data sources. Data consumers won’t (have to) know where the data is coming from.

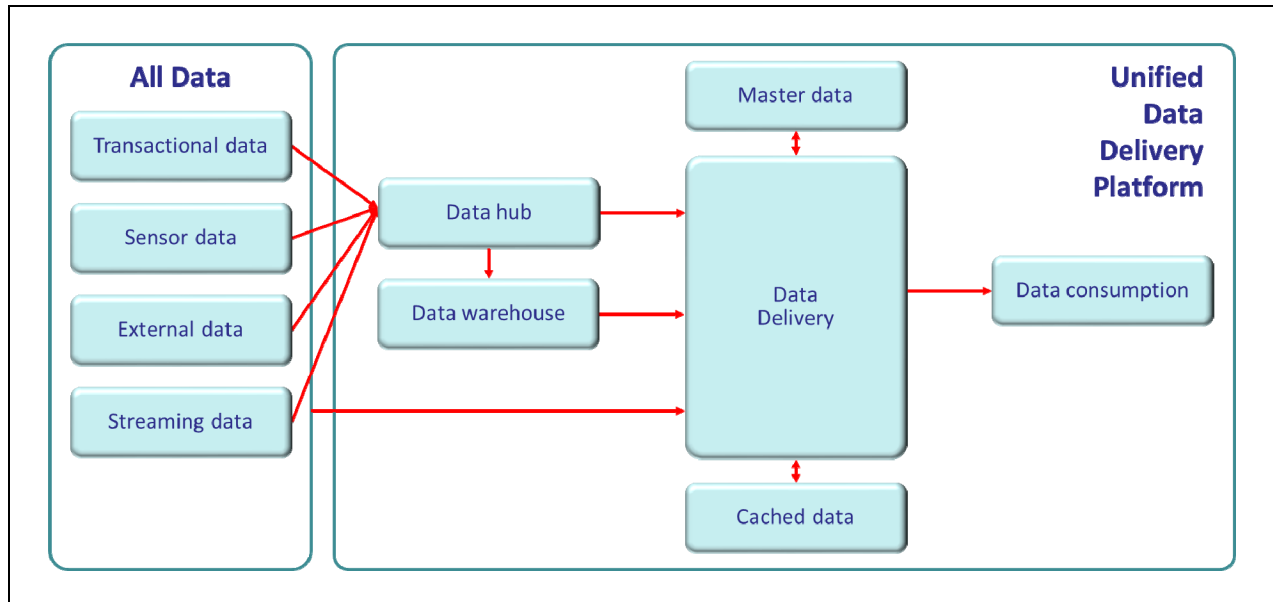


Figure 7 A high-level representation of the architecture of the unified data delivery platform.

For certain specifications *master data* is indispensable. Master data is used by all data consumers (without knowing it). *Cached data* is used by the data virtualization platform to preprocess and (temporarily) store the results of specific specifications. For example, if a data consumer needs data that requires complex and time-consuming calculations, the result of those calculations can be stored in the cache to speed up queries, or if specific data consumers require a stable view of the data for a long period of time, a cache may be defined as well.

The Different Forms of Data Usage – Different forms of data usage use different components of the architecture. For example, standard reporting and dashboarding use the components presented in dark cyan in Figure 8. This form of data usage uses the full capabilities of the virtual table layers, in other words, they may be accessing the top layer of virtual tables in which data is organized as a star schema to ease report development.

Another example of data usage is data science. Figure 9 illustrates the modules used by this form of data usage. To allow data scientists to access unprocessed data they may be accessing the data hub which contains a copy of the source data. Data scientists do not always access the data through the top layer of the virtual tables. Some prefer to access the original data. This implies that they access the bottom layer with virtual tables.

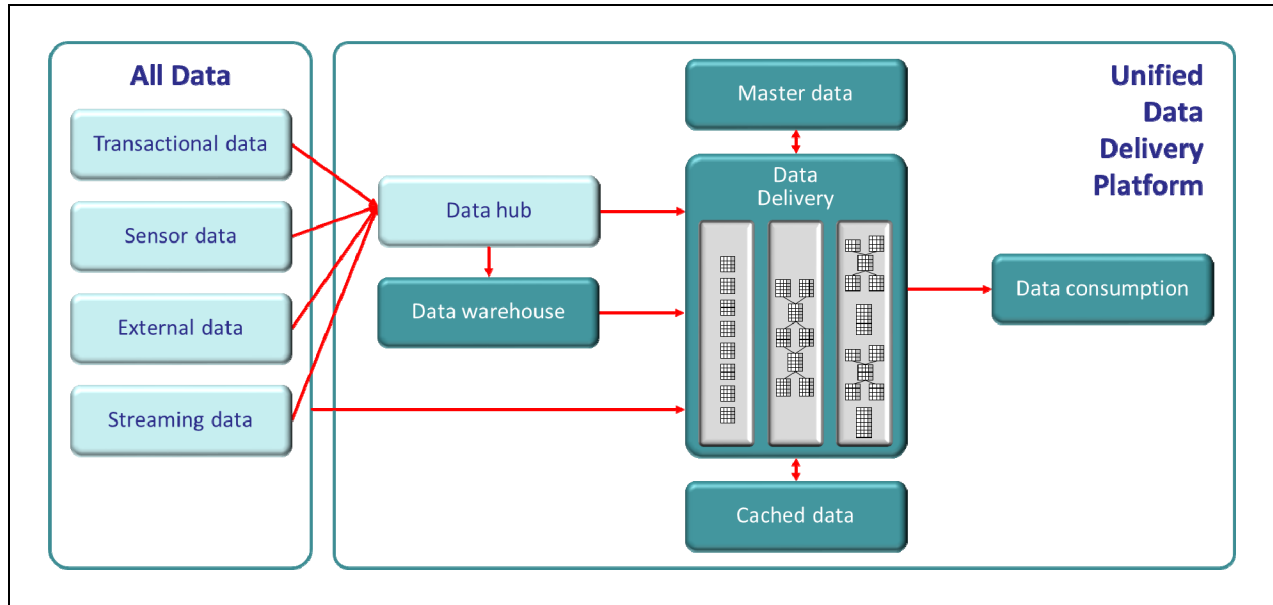


Figure 8 The components of the unified data delivery platform used by standard forms of reporting, analysis, and dashboarding.

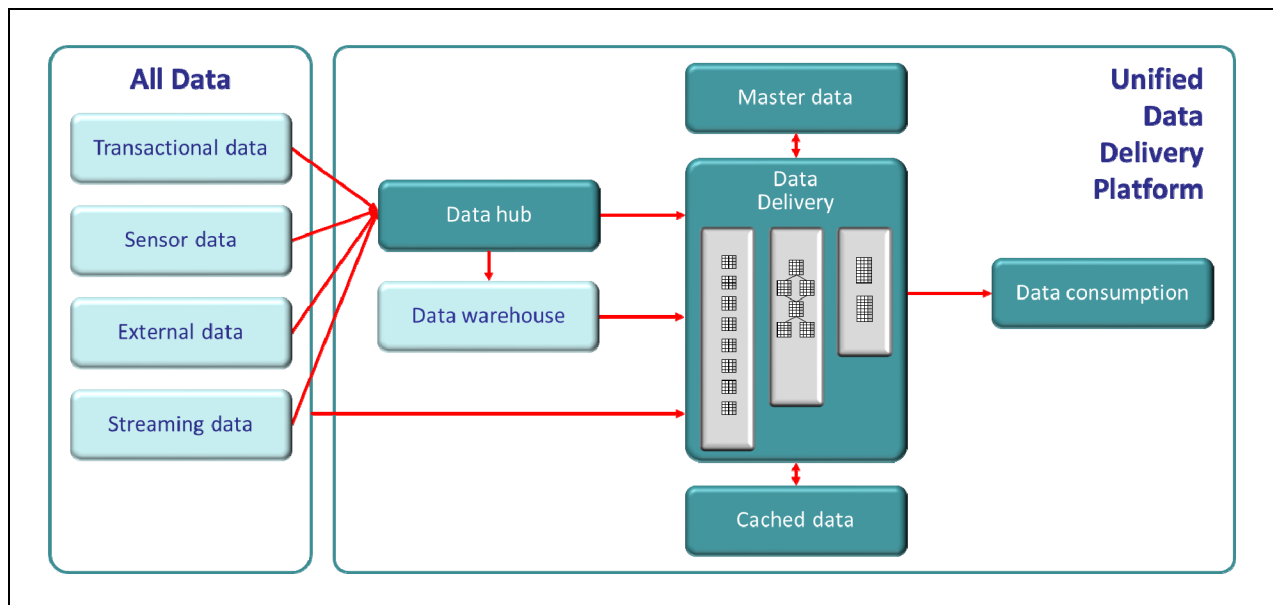


Figure 9 The components of the unified data delivery platform used by data scientists and other investigative users.

6 Requirements for Data Virtualization Technology

A Diverse Workload – The implementation of a unified data delivery platform must not only be able to support a large and complex workload, but also a diverse set of data usage forms. In fact, it must be able to support the combined workloads of all an organization’s data delivery systems, including:

- Reporting and analytics (from the data warehouse)
- Data science (from the data lake)
- Mobile apps (from the data services)
- Batch reports (from the managed file transfer environment)
- Real-time analysis (from the data streaming platform)
- Data product users (from the data marketplace)

No one-product silver bullet solution exists to implement a system capable of handling such a diverse workload. A multitude of products is required, including a database server, a master data management system, and a professional data cleansing product. However, one indispensable product type that forms the heart and backbone of the platform is a data virtualization platform. This technology is required for its data integration and abstraction capabilities.

A Diverse Workload – To be able to support a unified data delivery platform, a data virtualization platform must support at least the following features:

- **Scalable architecture:** Data virtualization platforms must be highly scalable. A scalable architecture is key for a unified data delivery platform, because, as indicated, it may have to support a large and diverse group of users: users coming from the traditional data warehouse environment, the data lake, the data marketplace, the mobile app environment, and so on. Technically this implies it must be able to parallelize the workload.
- **Elastic architecture:** The workload of a unified data delivery platform can fluctuate. For example, during the weekdays it may have to support many reporting and analytical users, while on weekends there may be a large workload coming in from mobile apps. Data virtualization platforms must be able to upsize or downsize the implementation dynamically without having to temporarily shut down the system.
- **Cloud-ready:** Predicting how successful the unified data delivery platform is and what the workload will be is difficult. Many organizations will therefore choose to run their unified data delivery platform in the *cloud* with its almost unlimited storage and processing capabilities. Data virtualization platforms must therefore support cloud solutions such as AWS, Google, and Microsoft Azure.
- **Big Data-ready:** This is the era of *big data*. More and more organizations opt for storing all the data they generate. They don't restrict themselves anymore to data that is used for reporting. Some even use the just-in-case approach for storing data. Data virtualization platforms must be able to handle massive amounts of data efficiently even if they are spread across many data stores and different types of data stores. Especially data scientists may require access to massive amounts of detailed data.
- **IoT-ready:** The *Internet-of-Things* is not something of the future. It's here and grows every day. The IoT consists of devices and applications sending messages with data to one another. This is the world of *data streaming*, or what some call *fast data*. A data virtualization platform must be capable of processing fast data. It must be able to receive streams of data, process it, and produce new streams.

- **Extensible architecture:** No data virtualization platform comes with all the bells and whistles required. For example, an advanced data cleansing feature is probably not supported, nor a complex calculation engine capable of specific financial calculation. Data virtualization platforms must somehow allow invocation of such external functional modules.
- **Multi-Key relationship discovery:** When new data sources have to be accessed, it's not always obvious what the data structures and the relationships are between tables of that data source and how those tables relate to the tables in the other data sources. Discovering all those relationships can be a time-consuming, tedious, and error-prone activity. Data virtualization platforms must offer support in simplifying this process of automatically discovering relationships. In this process it should not be restricted to discovering relationships based on single columns.
- **Centralized and fine-grained data security:** The more diverse the forms of data usage supported by a system, the more users with different access and authorization rules there will be. This requires data virtualization platforms to support extensive and fine-grained data security capabilities. Especially in this time of GDPR and other regulations related to data processing and storage, this is becoming increasingly important. A product never supports too much data security functionality.
- **API-rich:** Different users of the unified data delivery platform will use different tools and technologies to access the data. Therefore, a data virtualization platform must support a wide range of technical interfaces (APIs) from JSON/REST to JDBC/SQL.
- **Any data structure:** Not all data is stored in flat SQL table structures. Nowadays, data is stored in all kinds of forms and shapes. Some data is stored in non-flat structures such as hierarchical structures such as JSON and XML, and there is more and more data in the form of text, images, and video. A data virtualization platform must allow access to any data structure.
- **Advanced query optimizer:** The sophistication of the *query optimizer* of a data virtualization platform determines the performance of the overall system to a large part. The query optimizer must at least support query pushdown capabilities to delegate as much of the query processing to the underlying data source. This results in data processing close to where the data is stored. The query optimizer must also be able to execute as many components of a query in parallel. It must understand the strengths and weaknesses of the query optimizer of the underlying data source in detail to be able to generate the most efficient query for that data source. It must use a cost-based optimizer. It must be possible to study in detail how queries are executed to determine whether a better execution plan is possible.
- **Caching:** With caching (aka materialization) the virtual content of a virtual table is determined and physically stored in a data store. When a cached virtual table is queried, the stored cached data is accessed instead of the original data source. Caching can be used to minimize access of the data source and for offering a static query result content over a long period of time. A data virtualization platform must also support advanced refresh capabilities.
- **Detailed logging:** Every activity performed by a data virtualization platform must be logged and must be available for analysis. This is becoming increasingly important in these days of GDPR and other regulations related to data and privacy protection. It's also crucial for governance and auditing.

- **Data catalog:** Data virtualizations servers must allow elements, such as tables and columns, to be documented, defined and tagged. For developers, business analysts, and data scientists it's crucial that they can find the data they need. For example, when they need to develop a new virtual table or report they must be able to search for the right virtual tables. Such a search requires that all the existing virtual tables are cataloged and described in detail. Also, for business users who want to know what the data elements of the virtual table mean, it's crucial that they can find some descriptive information.
- **Master data management:** Wikipedia defines master data management as a method used to define and manage the critical data of an organization to provide [...] a *single point of reference*.² This is not always possible to enforce by using algorithms. In many cases, master data has to be stored and maintained in a separate system. Data virtualization platforms must support master data management functionality or must be able to integrate with master data management systems.
- **Data Cleansing:** Advanced cleansing features are required to get all the data on an acceptable quality level. Data virtualization platforms must support data cleansing functionality or must be able to integrate with such tools.
- **Data storage:** The unified data delivery platform has to store data temporarily or for a long period of time. Different reasons exist why data storage is needed. Some source systems don't keep track of history while some data consumers require this. It may be that access to a specific data source is slow or that the connection with the data source is unstable, demanding that that same data is stored in a separate data store that's part of the unified data delivery platform. For certain specifications some reference tables may have to be stored.
- **Forward writing:** Most of the workloads on a unified data delivery platform are query-oriented. Still, some applications may need to copy data from one data source to another through the data virtualization platform. This requires a data virtualization platform to support write functionality.

7 The fraXses Data Virtualization Platform

Introduction – *fraXses* is one of the newer data virtualization platforms on the market. It was introduced in 2015. The current version is called Leopard. It supports all the features expected from a data virtualization platform, including:

- It allows for *data federation* of a heterogeneous set of data sources, ranging from SQL sources via Hadoop and NoSQL to simple files and spreadsheets.
- It uses SQL-99 as the key language for defining all the transformations, filters, aggregations, joins, and cleansing operations to be applied to a data source. Most development is done through an intuitive, non-technical interface that generates data source specific queries.
- Data authorization rules can be defined centrally.
- It uses virtual tables as the key building block. These are called *data objects* in *fraXses*. These virtual tables can be accessed through different technical interfaces, including JDBC/SQL and

² Wikipedia, master data management, October 2018; see https://en.wikipedia.org/wiki/Master_data_management

JSON/REST.

- Dependencies between virtual tables can be presented graphically allowing for detailed *lineage and impact analysis*; see Figure 10.

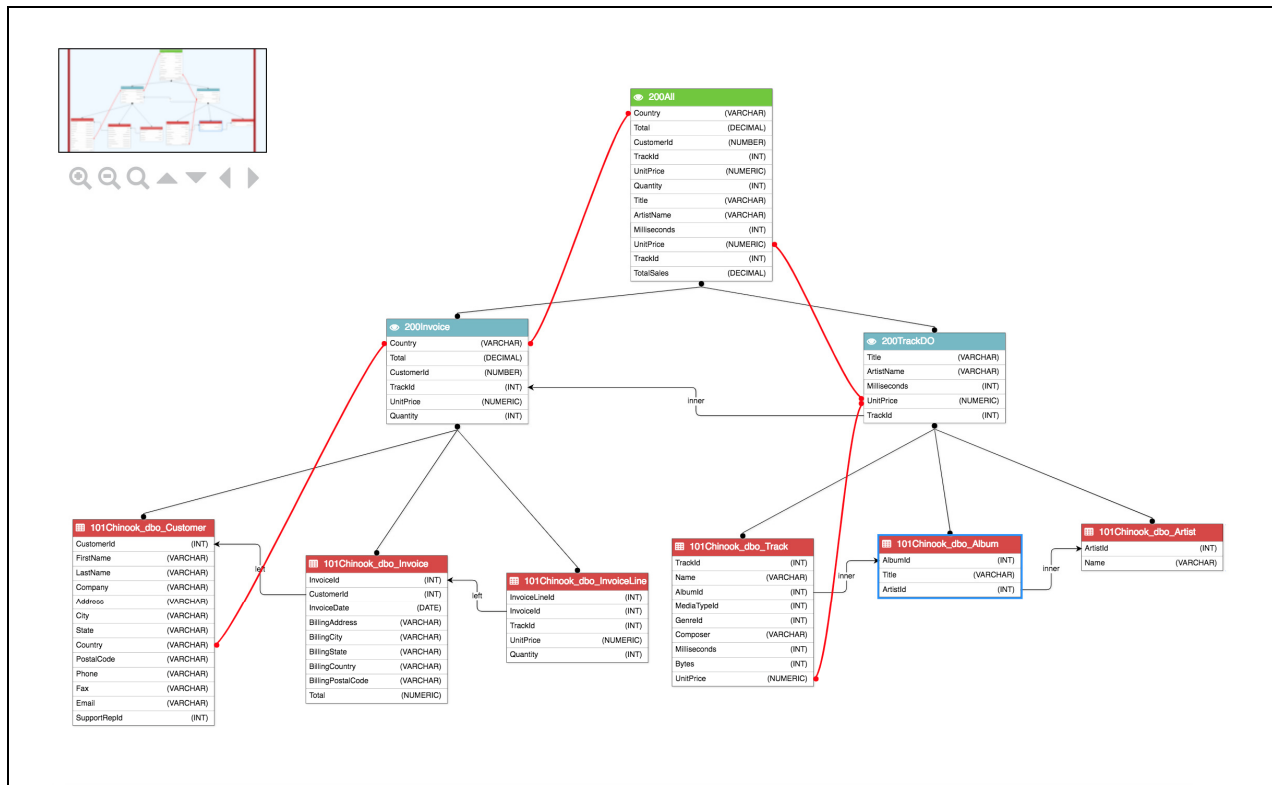


Figure 10 Detailed lineage dependencies are shown graphically using the red lines.

The product supports many unique features making it suitable for implementing a unified data delivery platform. These features are listed below.

The Microservices Architecture – Internally, fraXses is based on a modern and modular *microservices architecture*³. Each module is developed as a separate, independent service. Figure 11 shows some of the existing microservices making up the fraXses product.

For example, the JDBC service is used to access SQL-based data stores, the JSON service to access data stores using JSON, the query service optimizes incoming SQL queries and returns the most efficient query execution plan, the discovery service supports key discovery, the meta data service manages all the meta data on all the data sources, the scheduler service is responsible for invoking other services based on some time schedule, and so on. The *coordination service* is the traffic agent of fraXses. It’s responsible for coordinating communication between all the services. *Workload balancing* is one of the key features of the coordination service. For each service multiple instances can be started and stopped. Instances of the services can be spread across a cluster of nodes.

³ Wikipedia, Microservices, October 2018; see <https://en.wikipedia.org/wiki/Microservices>

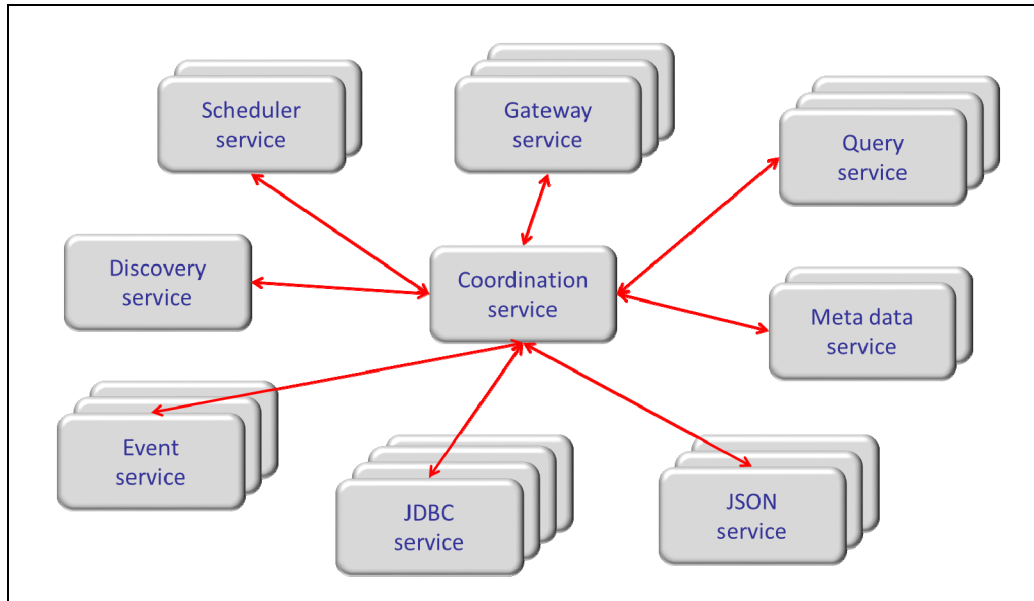


Figure 11 *fraXses* consists of a microservices architecture.

Scalable Architecture – There are many different ways to develop a microservices architecture. *fraXses* deploys a pattern called the *service mesh*⁴. This makes it a highly scalable architecture. Depending on workload requirements of each service, one or more instances can be started. For example, if the workload consists of many reporting users invoking new queries, multiple instance of the query service can be started, while if many business users are running similar queries (predefined reports), one or two of them may be sufficient. The setup of the services mesh can be adapted to the organization’s workload requirements. For example, in Figure 11 this is illustrated by showing four instances of the JDBC service and only one for the discovery service.

Elastic Architecture – With the coordination service the microservices setup is defined and can be changed dynamically. Through the coordination service this number can be increased or decreased depending on workload requirement changes. Stopping services does not require that the entire system must be shut down. If the number of instances is reduced, the coordination service stops directing new requests to that service instance. The service instance stops after completing the requests it was working on. So, the *fraXses* system can be upsized and downsized gracefully and dynamically, making it a complete elastic architecture. In other words, it can shrink and grow depending on the workload requirements. This allows an organization to set up an environment that is perfectly sized for their current workload. *Adaptive and elastic sizing* allows for creating the perfect balance between performance and scalability and costs. The screenshot in Figure 12 shows the services and instances at work.

Cloud-Ready – *fraXses* is available for on-premise and cloud environments, including AWS, Google, and Microsoft Azure. Organizations migrate from on-premise to cloud for various reasons, but one key reason is *elasticity*. If more processing or storage capabilities are required, in the cloud this can be made available in seconds. As indicated, the *fraXses* elastic microservices architecture is designed to support elasticity, therefore, it allows an organization to dynamically grow and shrink their *fraXses* environment depending on workload requirement changes.

⁴ F. Smith, What is a Service Mesh?, April 2018; see <https://www.nginx.com/blog/what-is-a-service-mesh/>

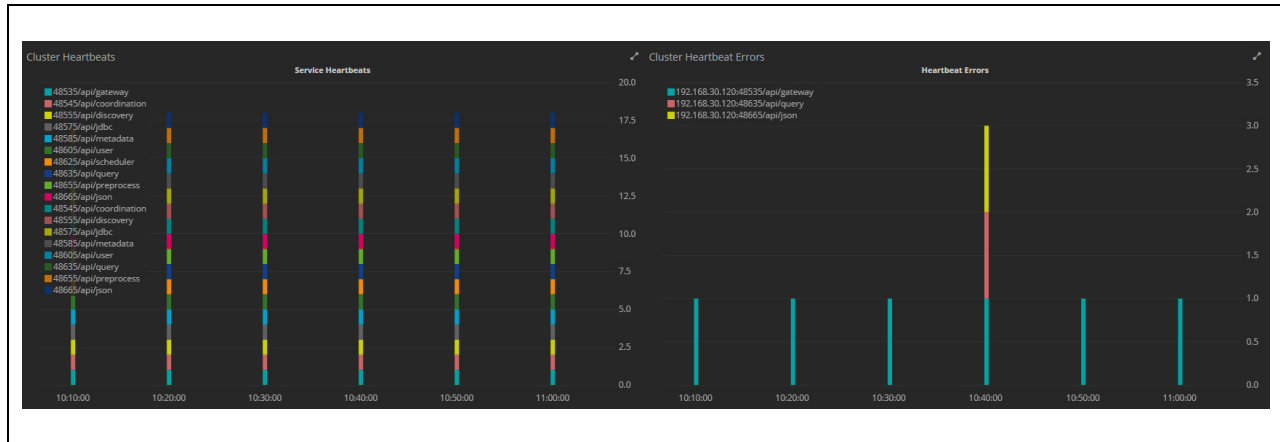


Figure 12 The services and instances of the fraXses data virtualization server at work.

Big Data-Ready – Due to its microservices architecture, multiple parallel instances of a service can be started. For example, when data has to be retrieved from Hadoop files, fraXses can start multiple instances of the right service on all the relevant nodes of the Hadoop cluster to execute the query in parallel. This allows massive files to be processed in parallel making the product *big data-ready*. The query optimizer is smart enough to understand that it should generate a query execution plan that makes use of those parallel service instance to access, for example, a Hadoop cluster.

IoT-Ready – fraXses supports data streaming (aka data pipelining). Messages coming from streaming technology such as Kafka can be pushed to the services of fraXses which can then process the messages and push them onwards to other services or applications. The Internet-of-Things consists entirely of devices and applications sending messages with data to one another. Due to the support for data streaming, fraXses can participate in that network. Virtual tables can be defined to process the incoming stream of data and send out transformed, aggregated, compressed or filtered data. In this case, fraXses acts as a complex-event processor. This is supported by the *FlowZ* module of fraXses.

Extensible Architecture – In the fraXses microservices architecture services communicate with each other. They send messages back and forth. To be able to this, each service has a standardized, open, and REST-based interface allowing, for example, the coordination service to talk to all the other services. This architecture is completely open. Others (third parties or customers) can extend fraXses with extra services that become first-class citizens in the fraXses architecture. For example, a home-made complex calculation engine or home-made data cleansing module can be added to the fraXses architecture allowing the other fraXses services to invoke them. The coordination service treats these new services as ordinary services, multiple instances of such services can be started, and their use is logged like all the other services.

Multi-Key Relationship Discovery – fraXses supports multi-key relationship discovery. It can discover single- and multi-key relationships between tables. Note that this module is not limited to discovery on SQL-based systems. Even the relationships with simple CSV files and Hadoop Parquet files can be discovered. Sources that use many columns with surrogate or artificial keys are handled smartly.

Centralized and Fine-Grained Data Security – fraXses allows centralized definition of rules for authorization and authentication. Authorization rules can be granted on a fine-grained level. Masking of data dependent on the users is also supported.

API-Rich – With fraXses multiple technical interfaces can be defined on a virtual table. For example, a JSON/REST interface can be defined to create a more service-oriented interface for Java apps, and a JDBC/SQL interface can be defined for users who come in using reporting and analytical tools, such as Microsoft PowerBI, QlikSense, and Tableau. When a virtual table is changed, the technical interfaces are automatically updated accordingly. No code has to be changed by hand.

Any Data Structure – With fraXses non-flat data structures can be imported and flattened. That data is presented to the users as flat data and can be accessed as any other SQL table.

Advanced Query Optimizer – The fraXses query optimizer supports query pushdown. Those parts of an incoming query (filters, projections, aggregations, joins) that can be processed by the underlying data store are pushed down to exploit the full power of that data store. Incoming queries that fraXses knows that are not processed efficiently by the data store are rewritten to more efficient queries before they are pushed down. The query execution plan generated by fraXses can be studied in detail. Statistical information on the data stores is gathered to fine tune the optimization process.

Caching – fraXses supports caching. The cached virtual tables can be stored in columnar Parquet files on disk, in SQL databases, and/or they can be kept in memory. Refreshing of the cached data can be scheduled. fraXses also supports real-time streaming into the caches..

Detailed Logging – Every activity performed by fraXses is logged. All this information is available for analysis and can be queried. Virtual tables can be defined on the logs to be studied by professional analytical tools.

Data Catalog – All the data elements, including virtual tables and columns, can be described.

Forward Writing – fraXses supports forward writing. Data can be copied from one or more data stores to another even if those data stores use different technologies. While copying, the data can be transformed, filtered, aggregated, and so on. So, the target data store doesn't have to have a comparable data structure to that of the source data stores.

Like most data virtualization platforms, fraXses doesn't support master data management and advanced data cleansing functionality, although the extensible architecture allows close integration with such features. A data catalog is not supported, although some technical meta data can be entered.

8 Closing Remarks

The biggest change in the IT industry of the last decade is related to how organizations use data. Organizations use data more intensively, more widely, and in many more forms. Organizations have started their digital transformation journeys and are striving towards becoming increasingly data-driven and this has led to this drastic change in data usage. Traditional forms of data usage, such as reporting and dashboards, are not sufficient anymore. There is a growing need to exploit the investment made in data by deploying almost every form of data usage, from simple reporting via mobile apps and self-service BI to data science.

Currently, to support all these new forms of data usage many organizations have developed a labyrinth of data delivery systems. This isolated approach of data delivery development has the following drawbacks:

- Decreased time to market of reports and analysis
- Decreased report consistency
- Duplication of data
- Decreased transparency
- Higher development costs

The *unified data delivery platform* supports a wide range of data usage forms. The key technology for such a unified platform is *data virtualization*. To support this new platform a data virtualization server must support minimally the following key features: scalable, elastic, cloud-ready, big data-ready, IoT-ready, and extensible architecture. Additionally, it must support multi-key relationship discovery, centralized and fine-grained data security, an advanced query optimizer, caching, detailed logging, and so on.

fraXses is one of the newer data virtualization platforms on the market. The product is based on a modern and modular *microservices architecture*. Each *fraXses* module is developed as a separate, independent service. The microservices architecture is scalable, elastic, cloud-ready, big data-ready, IoT-ready, and extensible. *fraXses* make it feasible to create a unified data delivery platform.

About the Author

Rick van der Lans is a highly-respected independent analyst, consultant, author, and internationally acclaimed lecturer specializing in data warehousing, business intelligence, big data, database technology, and data virtualization. He works for R20/Consultancy (www.r20.nl), which he founded in 1987.

He has presented countless seminars, webinars, and keynotes at industry-leading conferences. For many years, he has served as the chairman of the annual *European Enterprise Data and Business Intelligence Conference* in London and the annual *Data Warehousing and Business Intelligence Summit* in The Netherlands.

In 2018 he was selected the sixth most influential BI analyst worldwide by analytica.com⁵.

Rick helps clients worldwide to design their data warehouse, big data, and business intelligence architectures and solutions and assists them with selecting the right products. He has been influential in introducing the new logical data warehouse architecture worldwide which helps organizations to develop more agile business intelligence systems. He introduced the business intelligence architecture called the *Data Delivery Platform* in 2009 in a number of articles⁶ all published at B-eye-Network.com. The Data Delivery Platform is an architecture very similar to the logical data warehouse.

Over the years, Rick has written hundreds of articles and blogs for newspapers and websites and has authored many educational and popular white papers for a long list of vendors. He was the author of the first available book on SQL⁷, entitled *Introduction to SQL*, which has been translated into several languages with more than 100,000 copies sold. More recently, he published his book⁸ *Data Virtualization for Business Intelligence Systems*.

Ambassador of Kadenza: Rick works closely together with the consultants of Kadenza in many projects. Kadenza is a consultancy company specializing in business intelligence, data management, big data, data warehousing, data virtualization, and analytics. Their joint experiences and insights are shared in seminars, webinars, blogs, and white papers.

For more information please visit www.r20.nl, or send an email to rick@r20.nl. You can also get in touch with him via LinkedIn and Twitter (@Rick_vanderlans).

About fraXses, Inc.

fraXses is an innovative software company that believes in helping clients make the most of the value of their data while simplifying the process of data management.

⁵ [Analytica.com](http://www.analytica.com), *Business Intelligence – Top Influencers, Brands and Publications*, June 2018; see <http://www.analytica.com/blog/posts/business-intelligence-top-influencers-brands-publications/>

⁶ See <http://www.b-eye-network.com/channels/5087/view/12495>

⁷ R.F. van der Lans, *Introduction to SQL; Mastering the Relational Database Language*, fourth edition, Addison-Wesley, 2007.

⁸ R.F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann Publishers, 2012.

Using a Discover, Configure and Deliver methodology, the platform enhances data access and reduces the need for development with a low/no-code approach. This framework empowers the business to gain much greater value from their data.

fraXses provides an end-to-end solution for data virtualization and federation across multiple sources, technologies and locations as well as providing a data lake, IoT and data pipelining. The platform is built on a microservices architecture, which allows endless scalability options.

With the head office based in Welwyn Garden City, UK, fraXses supports global brands across broad commercial and financial sectors. For more information, visit www.fraxses.com or call +44 20 3012 5000 in the United Kingdom.